Replacing Oblivious Computation with Private Search for Context Sensitive Communications

Chaitrali Amrutkar, Rishikesh Naik, Italo Dacosta and Patrick Traynor
Converging Infrastructure Security (CISEC) Laboratory
Georgia Tech Information Security Center (GTISC)
Georgia Institute of Technology
chaitrali@gatech.edu, {rnaik6, idacosta, traynor}@cc.gatech.edu

ABSTRACT

Context aware applications provide users with an increasingly rich set of services. From services such as interactive maps to restaurant guides and social networking tools, the use of information including location, activity and time can greatly enhance the ways users interact with their surroundings. Unfortunately, the dissemination and use of such information also potentially exposes private information about the user themselves. In this paper, we present Themis, a framework for developing two-party applications capable of making decisions based on context sensitive information without revealing either participants' inputs. Themis uses private stream searching to replace the memory and computationally intensive oblivious computation associated with related techniques. We compare the security guarantees and performance profile of our approach against Fairplay and show not only as much as a 96% improvement in execution time, but also the ability to efficiently run applications with complex inputs on both desktop computers and mobile phones. In so doing, we demonstrate the ability to create efficient context-sensitive applications based on private searching.

1. INTRODUCTION

Context sensitive applications enhance interaction between users and their environments. From restaurant guides tailored to location and category to social networking and friend finders, these applications leverage information specific to individual users to dynamically deliver content or make decisions. These applications have become particularly popular on mobile devices, where context such as location and user activity change very rapidly.

Contextual information is often sensitive by its very nature. Location not only reveals a user's physical presence, but may also inform the observer of their activity (e.g., stadium, grocery store, adult entertainment complex). For applications including Google Latitude [15] that share such sensitive information with a potentially large number of other people, users are entirely reliant upon the correct implementation of policy in the application developer's servers for their privacy. Should these resources be compromised or the policies of the developer change, sensitive user data may needlessly be exposed. Secure Function Evaluation (SFE) addresses many of these issues and can be used to answer questions based on sensitive information without revealing the data itself or requiring the intervention of a third party. Unfortunately, these operations are computationally and memory intensive and have therefore not been widely explored for use in a constrained mobile setting.

In this paper, we develop *Themis*[1] an efficient means of answering context sensitive questions without revealing the potentially sensitive inputs used to make such decisions. Our approach differs from traditional SFE as it replaces expensive oblivious computation with more efficient private stream searching. In particular, we encode the context sensitive questions of an application as encrypted queries that operate on the receiver's data. Because of the homomorphic properties of the underlying encryption algorithms, our approach can be used to solve both direct comparison (e.g., which user is taller) and optimality (e.g., which restaurant is the mutually most favorable) questions without revealing the participants' inputs.

We make the following contributions:

- **Substitute Oblivious Computation with Private Search:** We demonstrate that computations in a context sensitive application can be transformed into search. We design a privacy preserving framework for matching and optimization based two-party applications using private search.

- **Implement and Characterize Themis:** We design, implement and measure the performance profile of Themis for both matching and optimization problems. Our analysis occurs using both a desktop and a G1 mobile phone running Android.

- **Compare Security and Performance Against Fairplay:** We show that Themis provides equivalent security properties as Fairplay under the honest-but-curious adversary model. We then demonstrate that Themis can perform significantly better than Fairplay, which is unable to run some applications with more than trivial inputs on the mobile phone.

The remainder of this paper is organized as follows: Section 2 presents an overview of important related work; Section 3 provides some definitions and preliminaries of the cryptography used in Themis; Section 4 describes the protocols for matching and optimization based applications; Section 5 offers the security guarantees of our framework; Section 6 gives the results of our performance analysis; Section 7 explores potential applications of Themis; Section 8 offers concluding remarks.

2. RELATED WORK

[1]Themis is the Greek goddess of blind justice and truth.

The functionality of an application can be significantly extended by considering a user's contextual information. Data including location, identity, activity and time [8] are already used to enhance web [7] and mobile [21, 3, 6] applications. As such information is frequently sensitive, there have been a number of frameworks proposed to preserve user privacy [9, 28, 2, 12, 13, 11, 1, 4] for such applications. Many of the available frameworks depend on techniques including k-anonymity or spatial cloaking, which often provide limited guarantees of privacy or require significant computational or communication overhead [18, 16]. Finally, some frameworks require the participation of a trusted third party [28, 2], which may not always be available. Accordingly, efficient solutions with strong guarantees are needed in this space.

Secure Function Evaluation (SFE) allows two parties to compute the result of a potentially sensitive function without exposing either participant's inputs. For instance, two millionaires seeking to determine who has greater total wealth can use SFE to answer this question without involving a trusted third party. While a number of publicly available software libraries now help developers implement the complex cryptographic functions underlying these functions [23, 29, 14, 22, 19], the significant computational and memory overheads associated with SFE have slowed its adoption in many domains.

Another popular means of obtaining user data privacy is Private Information Retrieval (PIR). PIR allows a user to retrieve data from an untrusted server without revealing the search query. Various implementations of PIR have been proposed in the past [20, 5, 10, 27]. PIR's search-only property and single-side privacy provision in two-party communications restricts its usage in context sensitive applications. Nevertheless, there have been attempts to build protocols for context aware applications using PIR where these properties are sufficient [17, 24]. The protocol proposed in this paper transforms the private search procedure into computation by using intelligent search queries and properties of the query such as size. We couple PIR with noise addition techniques to assure fairness and privacy to both parties involved in a communication.

3. DEFINITIONS AND PRELIMINARIES

In this section, we define and explain private searching in a two-party communication between Alice and Bob. We briefly review the Paillier and ElGamal cryptosystems underlying the Themis framework. Appendix A provides more details about the Paillier cryptograsystem.

3.1 Private Search Function Definitions

Private or blind searching allows Alice to execute search queries on Bob's data without revealing the queries. If Alice and Bob are communicating, Alice initially generates an encrypted query containing the search keywords she is interested in. She sends the encrypted query to Bob, who then runs a search algorithm on his data and finds the resulting records matching Alice's search keywords. This encrypted result is returned to Alice who decrypts them and finds the result to her query. The important property of the scheme is that Bob can execute the search procedure even though he has no information of the keywords in Alice's search queries. The private search scheme depends on homomorphic public key cryptography. Her private searching procedure on Bob's data consists of the following operations:

3.1.1 *Query Generation :* $QGen(T, F, Z, k_A^+)$

Alice runs $QGen(T, F, Z, k_A^+)$ to generate the encrypted query Q_A of size Z. The input parameters contain the table T, which consists of the records/keywords over which Alice wants to search. The mapping function F is used to align records in T and Bob's table. k_A^+ is Alice's public key. Alice sends the encrypted query Q_A to Bob along with the additional parameters F, Z and public key k_A^+.

3.1.2 *Private Search :* $PSearch(Q_A, F, Z, k_A^+)$

Bob maintains his own set of records/keywords in table T_b and performs a private search on T_b on behalf of Alice. Bob uses the same mapping function F on the entries in T_b to align his records' indices with Alice's table. Thus, an identical record in T and T_b maps to the same index in Q_A. Bob selects all the entries in Q_A which correspond to the records in T_b and finds the matching entries in T and T_b. The result of the search proceudre R and Q_A are both hidden from Bob, because they are encrypted with Alice's public key. Bob sends R to Alice.

3.1.3 *Reconstruction :* $Reconstruct(R, k_A^-)$

Alice extracts the matched keywords in T and T_b from R. She decrypts R with her private key k_A^- and reconstructs the output of the matching keywords. There have been several efforts in the past to build private searching frameworks [5, 25]. We will use the above definitions to explain the private search in the rest of the paper.

3.2 Cryptosystems in Themis

The cryptosystem used by Themis depends on the class of application to be built. Themis requires the use of public key cryptography with homomorphic and probabilistic encryption properties.

3.2.1 *Homomorphic Encryption*

An encryption algorithm E() is homomorphic if given E(x) and E(y) for plaintexts x and y, it is possible to compute E(x $\perp$ y) without decrypting E(x) and E(y), where $\perp$ can be addition or multiplication.

- Multiplicative homomorphism : An encryption algorithm is multiplicative homomorphic when E(x)·E(y) = E(x·y). ElGamal and RSA are examples of cryptosystems with multiplicative homomorphic properties.

- Additive homomorphism : An encryption algorithm is additive homomorphic when E(x)·E(y) = E(x+y). The Paillier and Benaloh cryptosystems are examples of additive homomorphic cryptosystems.

3.2.2 *Probabilistic encryption*

Probabilistic encryption is an important property required by certain public key cryptosystems. The adversary has access to the public key of the target user. When the adversary observes a ciphertext C encrypted by the target user, he can attempt a chosen-plaintext attack. If the encryption algorithm is deterministic, the adversary can quickly determine the contents of all of the entries in Q_A. To combat this attack, some public key cryptosystems use a random factor in the encryption ensuring that each plaintext maps into one of the large number of possible ciphertexts. Thus, given a plaintext p, there exist many different representations of $E(p)$.

The Paillier cryptosystem is a public key cryptosystem with probabilistic and additive homomorphic properties. ElGamal is a public key, probabilistic and multiplicative homomorphic cryptosystem. Both the cryptosystems satisfy the required properties of Themis. Based on whether additive or multiplicative homomorphic properties are required, we choose either the Paillier or ElGamal cryptosystem to implement protocols using Themis as explained in the next section.

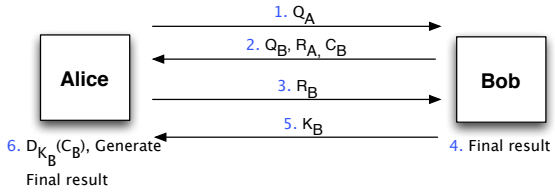

Figure 1: Matching based protocol : Information Flow

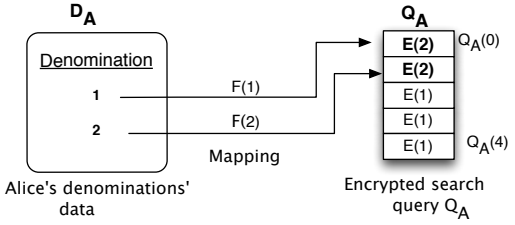

Figure 2: Matching based protocol : Query Generation at Alice for Millionaires' Problem application

4. PROTOCOL DEFINITION

The Themis framework is used to develop protocols, which preserve user data privacy in two-party context sensitive applications. It replaces the computations required in an application by private search. Being a search based framework, Themis requires the participants of an application to generate search tables containing their contextual data. Both the participants execute a private search operation on each other's data tables. To provide privacy to both participants involved in the communication, Themis introduces some noise to the search results. After processing the garbled search results, the initiator of the application exposes the output first. Themis introduces some fairness in the protocol by requiring both the participants to query each other and executing the protocol synchronously as shown in Figure 1 and Figure 6. Note that the search queries, results as well as the private search and noise procedures vary based on the application. Themis can be used to construct protocols for *two* types of applications : matching based and optimization based. In the following subsections, we will explain these two types of protocols with the help of example applications.

4.1 Matching Based Protocol

Matching based protocols can be used in applications where the output is binary (either TRUE or FALSE). An example of a matching based application is the Millionaires' Problem, where two millionaires wish to determine who is richer without disclosing their actual wealth. Themis can solve this problem by performing a matching procedure on the participants' search tables consisting of their wealth information. It executes the private search and noise addition procedures to output a 1 *bit* result depicting which one of them is richer. For the ease of understanding, we will use the millionaires' problem as a reference to explain the matching based protocols possible using Themis.

4.1.1 Query generation (initial setup) at Alice

Let us suppose that Alice has 2 million dollars and Bob has 3 million dollars. They build search tables containing their wealth. Let the wealth table of Alice be denoted by T_a and that of Bob by T_b. Thus, $T_a = \{1, 2\}$ and $T_b = \{1, 2, 3\}$. Let the size of Alice's query be 5. Alice generates a pair of public k_A^+ and private k_A^- keys using *ElGamal*.

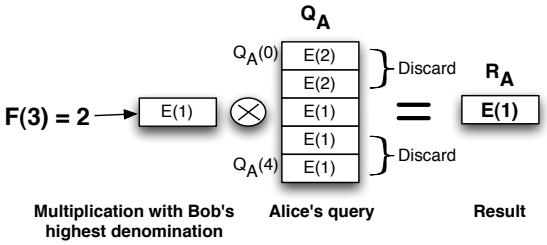

Figure 3: Matching based protocol : Private search at Bob's end in Millionaires' Problem application

Figure 2 depicts the query generation at Alice's end, when the size of query is $Z = 5$. Alice calls the algorithm $QGen(T_a, F, Z, k_A^+)$ to generate her query Q_A as explained in the background section. The function F maps each of the denominations in T_a to Q_A. Both Alice and Bob use F to align the entries in their data tables. Alice has a predefined function F_A for the millionaires' problem to assign values to entries in query Q_A.

for $i \leftarrow 1$ **to** Z **do**
  if $\exists\, d \in T_a : F(d) = i$ **then**
  / If some denomination maps to entry i, set i to* $E_{k_A^+}(2)$ **/*
    $Q_A(i) = \mathrm{E}_{k_A^+}(2)$
  else
    $Q_A(i) = \mathrm{E}_{k_A^+}(1)$
  end
end

If an entry in the wealth table is mapped to some index in the query, the value at that index is encryption of *two*. All the entries in Q_A, where no denomination maps, are set to encryption of *one*. Alice sends Q_A, F, k_A^+ and $Z = 5$ to Bob.

4.1.2 Private search at Bob

After receiving Alice's query, Bob performs $PSearch(Q_A, F, Z, k_A^+)$ on his data in T_b. Figure 3 represents the private search operation using Bob's function F_B.

Data: HD is the highest denomination in T_b. R_A is the result of the application.
for $i \leftarrow 1$ **to** Z **do**
  if $T_b(i) == HD$ **then**
  / If selected denomination is* HD*, generate* R_A **/*
    $x = F(T_b(i))$
    $R_A = Q_A(x) * \mathrm{E}_{k_A^+}(1)$
  else
    / Discard all the other entries in* Q_A **/*
    Discard $Q_A(i)$
  end
end

The function F_B selects the entry in Q_A corresponding to the highest denomination of Bob. It multiplies the selected entry with $\mathrm{E}_{k_A^+}(1)$ to get result R_A. It discards all the other values in Q_A. In Figure 3, Bob's highest denomination is 3. He calculates $F(3) = 2$ and multiplies the corresponding entry in Q_A with $\mathrm{E}_{k_A^+}(1)$. The result R_A is a single encryption value $\mathrm{E}_{k_A^+}(1)$ and it represents the

output of the application. The multiplicative homomorphic properties of ElGamal are exploited to obtain the output of the application by *only one* multiplication. Alice's query contains $\mathrm{E}_{k_A^+}(1)$ and $\mathrm{E}_{k_A^+}(2)$ values representing her wealth. Bob multiplies one of these values with an $\mathrm{E}_{k_A^+}(1)$. Thus, by the multiplicative homomorphic properties of ElGamal, the only possible values of R_A are either $\mathrm{E}_{k_A^+}(1)$ or $\mathrm{E}_{k_A^+}(2)$. Each of the output values correspond to the following:

- **1** : A '1' in R_A implies that Bob's highest denomination was not present in Alice's input query Q_A. The denominations at Alice and Bob are in ascending order. If Bob has a denomination in T_b which is not present in T_a, we can deduce that Bob is richer than Alice.
- **2** : A '2' in R_A implies that Bob's highest denomination was present in T_a. The denominations at Alice and Bob are in ascending order. If Bob has a denomination in T_b which is already present in T_a, it can be inferred that Alice is richer than Bob.

The standard Millionaires problem never provides any output if both Alice and Bob have the same wealth. This is because when the protocol outputs equal wealth, they get to know each other's inputs. Our aforementioned protocol sticks by the generic millionaires' problem. However, we can address the problem of equal wealth with a minor modification in the protocol.

Noise Addition : As already explained, the initiator of the application does not obtain the output of the application first. If R_A is sent back to Alice without modification, she will know the result of the application immediately and may not continue with the protocol execution. To avoid this, Themis adds noise to R_A. For noise addition in the millionaires' application, Bob generates an *even* random number n and encrypts n with Alice's public key. R_A is then multiplied by $\mathrm{E}_{k_A^+}(n)$ to obtain $\mathrm{E}_{k_A^+}(n * |R_A|)$[2]. Alice needs to know the output of the application sometime in the future, which requires her to know the value of noise added by Bob to obtain the solution. However, Bob has to make sure that he receives the output of the application from Alice first. To achieve this, Bob encrypts the noise factor and a timestamp using some symmetric encryption algorithm to get C_B as shown in Figure 1. C_B is Bob's commitment depicting that he added noise to the result at a particular time. C_B prevents Bob from sending a fake noise value to Alice at a later point and also ensures that Alice cannot get the result before Bob.

As already discussed, Themis requires both the participants to query each other's data for fairness. After completion of the private search operation, Bob generates a query Q_B using his public and private key pair k_B^+ and k_B^-. He creates the encryptions of $\mathrm{E}_{k_B^+}(1)$ and $\mathrm{E}_{k_B^+}(2)$ and executes $QGen(T_b, F', Z, k_B^+)$ using the algorithm in 4.1.1. Bob sends the new R_A value, C_B and his query Q_B to Alice.

4.1.3 Reconstruction at Alice

Alice decrypts R_A using k_A^- to get $(n * |R_A|)$. She cannot decipher the output of the protocol from this value as $(n * |R_A|)$ is always even and she knows that n is even. The plaintext of $|R_A|$ can be odd or even. To obtain the key to C_B and the result, Alice has to send output to Bob's query first. She runs $PSearch(Q_B, F', Z, k_B^+)$ algorithm to obtain R_B. She does not add any noise to R_B as Bob needs to obtain the output first to send her the key to decrypt the commitment C_B.

[2] $|R_A|$ = plaintext value of R_A.

4.1.4 Final result generation

When Bob receives R_B, he decrypts it to determine whether or not he is richer than Alice. After learning the output, he sends the symmetric key for commitment C_B to Alice. She decrypts C_B using the received key to obtain the noise value added to the plaintext of R_A. She divides the plaintext resulting from R_A by the noise value to finally obtain the output of the protocol. The next subsection runs through the millionaires' problem protocol using example values to prove its correctness.

4.1.5 Quick check example

Let Alice's wealth be $4\,million$ and that of Bob be $2\,million$. Therefore, initially Alice's table T_a contains $\{1, 2, 3, 4\}$ and Bob's table T_b contains $\{1, 2\}$. Let the size of the query table be $Z = 5$. Let the mapping function F map each of the denominations to an index *one* less than their integer value in the query table.

Alice's query Q_A consists of encryptions of $(2, 2, 2, 2, 1)$ in order. After reception of Q_A, Bob maps his highest denomination 2 to $Q_A(1)$. He multiplies $\mathrm{E}_{k_A^+}(2)$ at $Q_A(1)$ with $\mathrm{E}_{k_A^+}(1)$ to get R_A = $\mathrm{E}_{k_A^+}(2)$. Bob adds noise factor 8 to R_A to obtain $\mathrm{E}_{k_A^+}(8 * 2)$ = $\mathrm{E}_{k_A^+}(16)$. He then creates the commitment C_B by encrypting 8 with some symmetric encryption algorithm. Using the same mapping function F, Bob constructs query Q_B of size 5. Q_B contains encryptions of $(2, 2, 1, 1, 1)$ in order, encrypted using Bob's public key k_B^+. Bob sends Q_B, R_A and C_B to Alice. Alice maps her highest denomination 4 to $Q_B(3)$. She then multiplies $\mathrm{E}_{k_B^+}(1)$ at $Q_B(3)$ with $\mathrm{E}_{k_B^+}(1)$ to get R_B = $\mathrm{E}_{k_B^+}(1)$. Alice decrypts R_A to get the value 16. She cannot decipher the actual output by just using this value. Alice sends R_B to Bob who decrypts it to find that Alice is richer. On obtaining the output, Bob sends the key to the commitment C_B to Alice. Alice decrypts the commitment to get 8. She divides the plaintext of R_A that is 16 by 8 to get the final answer 2, which confirms that she is richer than Bob.

The synchronous information flow in the matching based protocol using Themis is depicted in Figure 1. Both Alice and Bob get the result of the application without knowing each other's exact wealth. Many other protocols for applications requiring binary results are possible using Themis using a similar scheme. We assume that Alice and Bob are semi honest users. We discuss the security assumptions and guarantees of Themis in Section 5.

4.2 Optimization based protocol

The matching based protocol can be used only in simple applications such as the millionaires' problem. Certain other classes of applications require an optimal solution generation over a set of data elements. For example, finding the *closest* Italian restaurant with the *highest* rating. The optimization based protocols built using Themis can be used in such applications. Let us consider another application example, where Alice and Bob need to set up an appointment at a mutually convenient Coffeebucks without knowing each other's locations. Initially, both participants are required to generate search tables containing the top n Coffeebucks locations geographically closest to them. However, only finding the common Coffeebucks locations does not lead to the optimal solution identifying the most convenient Coffeebucks. Therefore, the matching protocol is not sufficient for such scenarios. We will use this optimization application as a reference to explain the working of optimization based protocols. There are two steps in this particular optimization application. First, both Alice and Bob find the common Coffeebucks locations out of their individual lists of n. Secondly, Alice and Bob compute the optimal Coffeebucks location, where overall travel time is minimized.

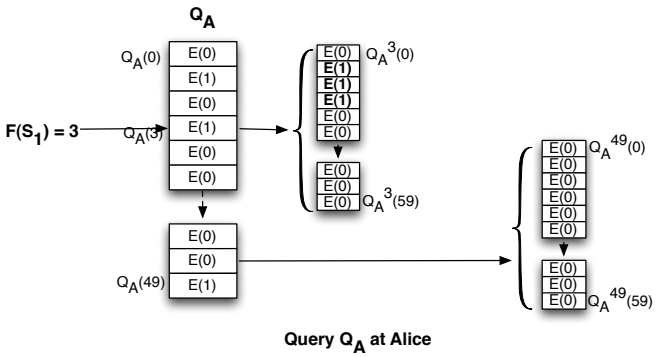

Figure 4: Optimization based protocol : Query generation at Alice for Coffeebucks application

Depending on the participants' individual schedules and perceived importance of the meeting, their willingness to wait for the meeting will be different. For example, if Alice is the CEO of a company and Bob is meeting her for a job interview, Alice may be ready to wait only for 15 minutes for the meeting to start. On the other hand, Bob would be willing to wait for 45 min. Let us call this value as individual wait time w. Let T_a and T_b be the initial data tables at Alice and Bob respectively with entries $(a_1, a_2..a_n)$ and $(b_1, b_2...b_n)$. The n entries correspond to the top n geographically closest Coffeebucks to each Alice and Bob. Each entry contains the location of a Coffeebucks, time to reach the Coffeebucks and A_w or B_w. For example, if Alice can reach S_1 in *seven* minutes and is ready to wait for the next *eight* minutes at S_1 for Bob to arrive, then $a_1 = \{S_1, 7min, 15min\}$. In the following subsections, we will explain the Coffeebucks application result generation when Alice queries on Bob's data. Bob follows exactly the same procedure with a different set of keys.

4.2.1 Query generation (initial setup) at Alice

Alice generates a pair of public k_A^+ and private k_A^- keys using the *Paillier* cryptosystem. Figure 4 depicts the query generation at Alice's end. We call each of the Z entries in Q_A a 'block entry' representing a Coffeebucks location. Each block entry consists of Z' subentries representing the time parameter of the application. Alice executes $QGen(T_a, F, Z, k_A^+)$. Each Coffeebucks location record in T_a is mapped to Q_A using function F to align it with Bob's data during the private searching step. Alice holds a predefined function F_A specific to the Coffeebucks application, to populate the entries in Q_A. Let $Q_A^v(u)$ denote the u^{th} subentry in block v in Q_A. There are two rules in F_A, which determine the values in the subentries in Q_A.

If Alice can be available at the Coffeebucks corresponding to block entry Q_A^v at time u, the subentry $Q_A^v(u)$ contains $\mathrm{E}_{k_A^+}(1)$. Alice can be available at the concerned Coffeebucks after her transit time and before the expiration of her wait time A_w. All the other subentries in the the query table are set to $\mathrm{E}_{k_A^+}(0)$. Consider the example in Figure 4. $Z = 50$, $Z' = 60$ and $n = 10$. Each subentry represents *one* minute, thus $Z' = 60$ represents an hour from the current time. Coffeebucks S_1 is mapped to block *three* using F. Alice takes *two* minutes to reach the coffee shop and can wait for *two* minutes after arrival. Therefore, the subentries $Q_A^3(1)$ and $Q_A^3(3)$ are set to $\mathrm{E}_{k_A^+}(1)$. The other subentries in Q_A^3 are assigned $\mathrm{E}_{k_A^+}(0)$. The entire table containing $Z * Z'$ subentries in Q_A is filled using the above rules. Each subentry is either $\mathrm{E}_{k_A^+}(0)$ or $\mathrm{E}_{k_A^+}(1)$. Due to the probabilistic encryption property of Paillier, the ciphertext $E(p)$ can have several different representations for one plaintext p. This property eliminates the possibility of an adversary correctly guessing the plaintext corresponding to a target ciphertext. Alice sends $\{Q_A, F, Z, k_A^+\}$ to Bob.

Data: $Q_A^v(u)$ represents the u^{th} subentry in block v in Q_A. T_a is the table containing n Coffeebucks location records for Alice. A_w is the total wait time for Alice. $T_a(m).ttr$ is the time to travel to the m^{th} Coffeebucks location record in T_a. $n << Z$.
for $i \leftarrow 0$ **to** Z **do**
  if $\exists\, T_a(m) : F(T_a(m)) = i$ **then**
  / If some Coffeebucks location maps to block i */*
    for $j \leftarrow 0$ **to** Z' **do**
      if $j + 1 \leq T_a(m).ttr \parallel j + 1 > A_w$ **then**
      / Set subentries before travel time and after wait time to $E_{k_A^+}(0)$ */*
        $Q_A^i(j) = E_{k_A^+}(0)$
      else
        / Set subentries corresponding Alice's availability at m to $E_{k_A^+}(1)$ */*
        $Q_A^i(j) = E_{k_A^+}(1)$
      end
    end
  else
    for $j \leftarrow 0$ **to** Z' **do**
    / If no Coffeebucks maps to i, set all subentries to $E_{k_A^+}(0)$ */*
    $C =$
      $Q_A^i(j) = E_{k_A^+}(0)$
    end
  end
end

4.2.2 Private search at Bob

Search : Bob performs $PSearch(Q_A, F, Z, k_A^+)$ on his data using the information received from Alice as shown in Figure 5. Bob's context information table T_b contains top n Coffeebucks locations close to Bob, his time to reach them and his waiting time B_w. Bob runs the predefined function F_B to generate the private search result as follows:

Data: T_b is the table containing n Coffeebucks location records for Bob. B_w is the total wait time for Bob. $T_b(m).ttr$ is the time to travel to m^{th} Coffeebucks location record in T_b. $R_B[n]$ is the array of result values. $n << Z$.
for $i \leftarrow 0$ **to** n **do**
  $F(T_b(i)) = pos$
  $j = T_b(i).ttr$
  $c = Q_A^{pos}(j)$
  while $j \leq B_w$ & $j \leq Z'$ **do**
    $c = c * Q_A^{pos}(j)$
    $j = j + 1$
  end
  $R_B[i] = c$
end

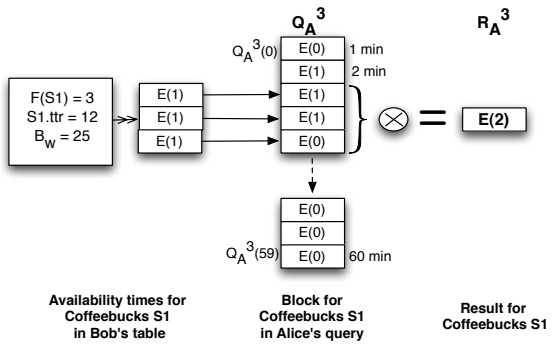

Figure 5: Optimization based protocol : Private Search at Bob for Coffeebucks application

Bob aligns his Coffeebucks location records in T_b to Q_A using function F received from Alice. For the meeting to take place, both Alice and Bob have to be available at a certain Coffeebucks at the same time. Bob does not have information about Alice's availability because her query is encrypted. However, he knows that their inputs are aligned because of function F. Therefore, Bob can obtain the correct output by only inputing his availability for searching on Q_A and finding overlaps with Alice's inputs. To achieve this, Bob selects subentries from Q_A corresponding to the times at which he can be available at the Coffeebucks locations in his table. Following the query generation process, Alice's availability is represented by an $\mathrm{E}_{k_A^+}(1)$. Therefore, if Alice has any intersecting times for certain Coffeebucks location with Bob, some or all of the selected subentries for that Coffeebucks will be $\mathrm{E}_{k_A^+}(1)$. If Bob selects an entry $Q_A^v(u)$ containing $\mathrm{E}_{k_A^+}(1)$, it implies that both Alice and Bob can be present at Coffeebucks v at time u from the current time. If Bob selects an entry $Q_A^v(u)$ containing $\mathrm{E}_{k_A^+}(0)$, it implies that Bob can be present at Coffeebucks v at time u from the current time and Alice cannot. This may be because either Alice's table does not contain Coffeebucks v or she cannot be available at v at time u.

In case of multiple matches of Coffebucks in T_a and T_b, Bob will select $\mathrm{E}_{k_A^+}(1)$ subentries from Q_A for more than one Coffeebucks locations. He will then require to calculate the optimal Coffeebucks location which minimizes his and Alice's transit time. We leverage the fixed wait times A_w and B_w to arrive at the optimal solution as follows :

Bob executes the selection procedure on each block in Q_A corresponding to the entries in T_b. After the completion of the selection procedure, Bob multiplies all the values selected per block entry to get a *single* encrypted result entry for each block corresponding to entries in T_b. Bob obtains n results after the selection and multiplication operations as the size of T_b is n. We claim that the result entry containing ciphertext of the *highest plaintext value* is the optimal solution. Let us take an example to prove the correctness of this claim.

Consider a block v in Q_A corresponding to Coffeebucks S_v. The wait times A_w and B_w represent a participant's willingness of waiting for the combined time of transit to S_v and waiting time at S_v before the other party arrives and the meeting begins.

$\forall\ S_a \in T_a$: Transit time to S_a + availability time at $S_a \leq A_w$
$\forall\ S_b \in T_b$: Transit time to S_b + availability time at $S_b \leq B_w$

Alice's availability at S_v is denoted by $\mathrm{E}_{k_A^+}(1)$ in the subentries corresponding to her available times. Let $F(S_v) = v$. An $\mathrm{E}_{k_A^+}(1)$ in a subentry $Q_A^v(u)$ implies that Alice has already arrived at S_v and is waiting for Bob. Using the above equations we can infer that the transit time for Alice to S_v is less than or equal to u. If a block v in Q_A contains the maximum number of $\mathrm{E}_{k_A^+}(1)$ values, we can infer that Alice is available for the maximum time at S_v. Given that her fixed total wait time A_w is the summation of her transit time and her available time, maximum available time at S_v implies minimum transit time to S_v as compared to other Coffeebucks. This simple observation can lead us to the optimal solution.

The optimal solution corresponds to the Coffeebucks to which both Alice and Bob have minimum transit times and are therefore able to begin the meeting at the earliest. Thus, the block in Q_A corresponding to a Coffeebucks, where the intersection of the availability time of Alice and Bob is the maximum will be the solution. This indicates that the block in Q_A, which has the maximum number of $\mathrm{E}_{k_A^+}(1)$ (available times) after Bob's selection procedure is complete, corresponds to the optimal solution. If Bob multiplies all the selected entries from each block, he will get the ciphertext of the addition of all the plaintexts (additive homomorphism). The block containing the maximum number of $\mathrm{E}_{k_A^+}(1)$ will produce ciphertext with the maximum plaintext value and will correspond to the optimal solution. Therefore, Bob executes matching and multiplication operations on Alice's query entries to obtain n blocks in result R_A. The maximum valued entry in R_A is the optimal solution for the application.

Noise addition using result sampling :
Several privacy concerns arise if Bob sends R_A to Alice immediately after executing private search. Any entry in R_A containing a non-zero value corresponds to a matched Coffeebucks location. Alice can maintain the mapping of the Coffeebucks locations in T_a to Q_A using F at the time of query generation. If Bob sends result R_A to Alice as it is, Alice can decipher which Coffeebucks locations matched after decrypting R_A and doing a reverse lookup in the mapping table. Moreover, by analyzing the result of a matched block entry in R_A and the values in her corresponding block entry in Q_A, Alice may be able to infer Bob's availability. The knowledge of Bob's availability times at multiple Coffeebucks can allow Alice to triangulate Bob and estimate his actual location. For example, if Alice sends *five* $\mathrm{E}_{k_A^+}(1)$ subentries in a block entry and gets a *three* in the corresponding result entry in R_A, she can infer Bob's time preferences.

In some situations, even if Alice cannot guess the matched Coffeebucks locations, just knowing the total number of Coffeebucks matches can leak some information to her. This problem will occur when Alice and Bob are geographically very close to each other and thus all or most of the locations in T_a and T_b match. For example, if Alice queries for *five* Coffeebucks locations and after decryption she gets *four* non-zero block entries, she can deduce that *four* out of *five* Coffeebucks locations match with Bob. Even though she does not know which ones matched, the number of available possibilities of picking *four* out of *five* Coffeebucks is only $\binom{5}{4} = 5$. She will have to try only these *five* combinations to triangulate Bob. Finally, sending R_A without any modification allows Alice to query Bob privately and get results of the application before Bob. Themis does not allow the initiator of an application to get the results first.

To avoid these security issues, Bob adds noise to R_A before sending it to Alice. The aim of noise addition is to prevent Alice from triangulating Bob and to maintain fairness by providing Bob with some control over Alice's queries. However, the optimal so-

lution in the result must not be lost after noise addition. The noise addition should only preclude Alice from obtaining the final result, while allowing her to correctly deduce the intermediate optimal solution after decryption. Therefore, Bob uses result sampling during the private search procedure to add noise. We will explain the noise addition part with the help of Figure 5:

1. Bob selects the subentries in Alice's table corresponding to his availability for each Coffeebucks, while executing private search. He then multiplies all the subentries to generate the result for the block. Let us suppose that Bob picks entries $(k1, k2, k3)$ from block T and their multiplication gives $\mathrm{E}_{k_A^+}(k)$. Let $\mathrm{E}_{k_A^+}(k)$ correspond to the optimal solution. If Bob can conform the plaintext value of noise entries to less than k, the optimal solution will not be lost. Addition of random entries may mislead Alice to select the incorrect maximal value as the optimal result. Moreover, if the random values look very different from the expected output, Alice can differentiate between noise and real data with probability 1. Therefore, Bob samples the candidate entries for multiplication in Alice's table and adds them back as noise. He picks only x of the actual candidates, where x is less than or equal to the total entries chosen in that block. He multiplies these noise candidates together to get a value which is definitely not greater than the total value of multiplication candidates. For example, in Figure 5, the candidate values for multiplication are $(k1 = 1, k2 = 1, k3 = 0)$. The total sampling options possible are $(k1, k2, k3, k1{*}k2, k2{*}k3, k3{*}k1, k1{*}k2 * k3)$. The different possible values for the noise will be between $\mathrm{E}_{k_A^+}(0)$ and $\mathrm{E}_{k_A^+}(2)$. Even if any one of these sampling options is chosen, it never has greater value than the optimal solution. Let us call the noise value $noise_{val}$.

2. Bob creates a new entry at the end of R_A and copies the $noise_{val}$ into the new entry. He multiplies the new entry with $\mathrm{E}_{k_A^+}(0)$ to ascertain that the ciphertext $noise_{val}$ appears different than the ciphertexts used for its generation. This procedure helps in preventing a co-relation attack on the result by Alice.

3. Bob repeats the first two steps n times, one each for the Coffeebucks location records in T_b during private searching. The result table size after noise addition is $2n$.

4. Bob shuffles all the result entries in R_A to prevent Alice from simply discarding the noise values added to R_A after the first n results. He maintains the mapping of the original and new indices which he uses for final result generation in section 4.2.4.

Bob sends R_A to Alice.

4.2.3 *Intermediate optimal result at Alice*

After Alice receives R_A from Bob, she decrypts it using k_A^- to obtain a table containing integer values. She knows that a subset of the non-zero values corresponds to matched Coffeebucks locations. However, due to the addition of noise in the result, Alice cannot differentiate between the exact matched Coffeebucks locations and the false ones. Furthermore, she cannot determine the total number of matched Coffeebucks locations because the noise addition can add a *zero* or a *non-zero* entry to the result. As explained in Section 4.2.2, the optimal solution corresponds to the maximum value in the result table. Thus, Alice simply selects the entry which has the maximum plaintext value and sends its index information back to Bob.

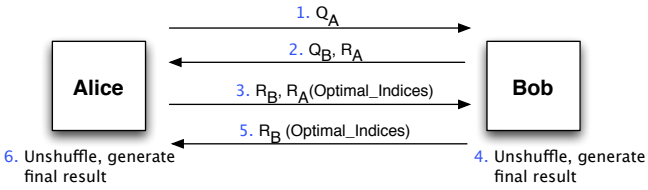

Figure 6: Information flow in the optimization based protocol with Themis

4.2.4 *Final result generation*

Bob matches the index of the optimal solution with the already saved mapping data during shuffling to find the original index of the result entry. Because of the noise addition or in some cases legitimately, there can be more than one indices of optimal solution. When Bob gets the indices, he discards the ones which were added as noise using the mapping maintained during noise addition. If he is left with more than one solutions after that, he uses some predefined tie-breaker algorithm based on street names or any other parameter to finalize the optimal solution. This algorithm has to be known to both Alice and Bob. Bob determines the optimal Coffeebucks location $S_j \in T_b$ by doing a reverse mapping using function F.

In the optimization based protocol, all the above steps are performed from Bob's end as well. Bob generates a pair of public and private keys k_B^+ and k_B^-. He formulates query Q_B using T_b and runs it on Alice's data in T_a. Alice finds R_B and sends it back to Bob. Bob decrypts R_B, computes the optimal solution and sends the solution to Alice. The synchronous information flow in the optimization based protocol is depicted in Figure 6. After both Alice and Bob obtain the final result, both of them travel to the resulting Coffeebucks location. They possess no information of when the other party will arrive. However, the protocol guarantees that the meeting begins before the lesser of their time to wait (A_w or B_w) expires.

The example applications mentioned in this section were used for the ease of understanding the protocol creation. Appendix A offers details of more example applications. All the operations in a protocol built using Themis change depending on the class of the application using it. However, the basic concept of using private search and noise to provide user privacy remains the same across all applications. Because the concepts of building the protocols remain the same, the security guarantees provided by Themis are equivalent for all protocols as described in the next section.

5. SECURITY ANALYSIS

5.1 Assumptions and Security Guarantees

Themis prevents exposure of context sensitive data in two-party communications. The framework uses existing concepts of private searching coupled with noise addition procedures to generate efficient privacy preserving protocols. The Themis framework provides security guarantees based on the following assumptions:

- The participants are *honest-but-curious*. Participant may log all the data exchanged during the course of the application but execute the protocol correctly. We assume that the participants do not alter the data in their search tables during the protocol execution.

- Private search using both ElGamal and Paillier cryptosystems is secure. Private searching scheme guarantees that a client can search a server's data privately without the server determining the contents of the query or its results. We also assume that AES is a secure block cipher when used for generating commitment value in the matching based protocol.
- Both participants have polynomially bound computational resources.

The security guarantees provided by Themis model are :

- Both Alice and Bob are guaranteed that the other party will not learn more information about them than what is exposed through the output. For example, in the millionaires' problem, both parties learn their wealth relative to the other participant, but never the actual value of the other participant's wealth.
- The probability that Alice or Bob can cheat is negligible because of our assumption regarding the security provided by PIR schemes.
- In any application protocol built using Themis, the participant who does not initiate the application always gets the results first. Like Fairplay, we do not provide fair termination. If the receiving participant quits before sending the final message to the initiator, the initiator can not complete the protocol successfully. No simple solutions exist to address this issue.

When Alice initiates a Themis protocol, Bob adds noise to the result of Alice's blind query to ensure that he receives the output of the application first. This noise addition must ensure that Alice can guess the output with a very small probability or with the same probability before the start of the protocol. We discuss the probability of Alice guessing the result after noise addition in each of the sample applications:

5.1.1 Matching application

The millionaires' problem has two possible solutions: either Alice is richer or Bob is richer.[3] The probability of Alice guessing the correct output is therefore $0.5 + \epsilon$. Let the plaintext of Alice's private search result on Bob's table be n_1 and the plaintext of the noise added by Bob be n_2. If n_2 is always even, the output of the multiplication of n_1 and n_2 will always be even irrespective of whether n_1 is odd or even. By solely observing only $n_1 * n_2$, Alice can tell with $0.5 + \epsilon$ probability whether n_1 is odd or even. Our protocol therefore does not increase Alice's probability of guessing the result even after receiving the garbled output of her private search. As Alice can not determine the plaintext values corresponding to Bob's query, the decrypted response that Bob receives will be correct assuming that Alice has entered her wealth truthfully.

5.1.2 Optimization application

For simplicity, we assume that there is only one possible optimal solution for the private search optimization problem. Let n be the number of Coffeebucks in Alice and Bob's data tables. When Alice initiates the protocol, her probability of guessing the optimal solution in case of a match is $\frac{1}{n} + \epsilon$. After receiving Alice's encrypted query, Bob executes a private search on his data on her behalf to get result R_A. To prevent Alice from retrieving the output of the application first, Bob adds noise to R_A. As discussed in Section 4.2.2, Bob adds noise to R_A by sampling the subentries in Alice's table. The subentries selected for sampling represent the times at which he can be present at the corresponding Coffeebucks. Bob multiplies all the selected subentries to determine the total time for which both him and Alice can be present at the Coffeebucks. Let us suppose that Bob picks subentries $(k_1, k_2 \cdots, k_m)$ from block Q_A^X. The Coffeebucks corresponding to X may or may not be present in Alice's query. If it is not present, Alice will have *zero* $\mathrm{E}_{k_A^+}(1)$ in block X and therefore the sampling will always give a zero. However, if there is an overlap between Alice and Bob's preferences in block X, the total time for which both of them can be present at the concerned Coffeebucks may be non-zero. This result would be obtained by $\prod_{n=1}^{m} k_i$. While calculating $\prod_{n=1}^{m} k_i$, Bob also creates samples of the result to generate noise values. The total number of samples S possible over $(k_1, k_2 \cdots, k_m)$ in Q_A^X is:

$$S = \sum_{i=1}^{m} \binom{m}{i}$$

The value in each of $(k_1, k_2 \cdots, k_m)$ can be either a *zero* or a *one*. The range of possible plaintext values for noise after combining each of the additive homonorphic ciphertexts in a sample would be from *zero* to $\sum_{i=1}^{m}(k_i)$. After the noise addition for each result block, Bob shuffles the entries to get an R_A of size $2n$ and sends the result to Alice. When she decrypts R_A, she is unable to identify the actual matches and even the total number of matches over the noise. She only sees an array of *zero* and non-zero values. However, Alice can not filter out any of the values from R_A as all appear to be legitimate. This is because the noise can range from *zero* to $\sum_{i=1}^{m}(k_i)$. Alice can be certain that the maximum value of the plaintexts returned does represent the optimal value given the sampling method and can therefore return the corresponding index to Bob who in turn can learn the optimal location.

5.2 Comparison with Fairplay

Fairplay [22] is an existing implementation of Secure Function Evaluation (SFE). SFE allows two honest-but-curious participants to perform computation on their private data without necessarily revealing it. In Fairplay, only one of the two parties generates the required circuits and keys. The other party performs only circuit evaluations and is completely oblivious to the evaluation operations. This property makes the Fairplay protocol unfair to one of the users as it gives additional advantage to the user who generates the keys and circuits. To prove this point, let us assume that Bob generates all the keys and circuits, and Alice evaluates the circuits. Alice initializes the application by requesting for m circuits from Bob. She then selects *one* circuit out of m to evaluate the final output. This step allows Bob to lie with a probability of $\frac{1}{m}$. However, because all the circuits are oblivious to Alice, she can lie with negligible probability. Thus, Fairplay is generally unfair to the initializer of an application. In comparison, both Alice and Bob execute the same blind search operations on each other's data in Themis and can lie with negligible probability under an honest-but-curious model. The approach of both participants executing the same operations provides fairness in Themis. Both the participants need comparable amount of memory and processing power for an application execution. In Fairplay, the generator of keys and circuits incurs more processing and memory overhead than the evaluator.

As discussed in the protocol section, Themis generates similar information leakage to Fairplay. For matching applications such as the millionaires' problem, only a single binary bit is output. Optimization applications expose only the optimal solution to the participant and conceal all other response values with noise.

The security guarantees provided by Themis are comparable to

[3]The case in which their wealth is equal is generally not considered as this would reveal the inputs to the participants

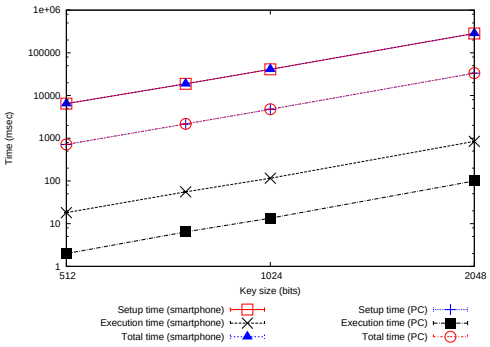

Figure 7: Themis' performance for the millionaires' problem application for different key sizes and constant table size (500 rows) in the PC and the smartphone. As expected, the use of larger keys increases the time required by the application. Also notice that the total time and the setup time are very similar for both platforms.

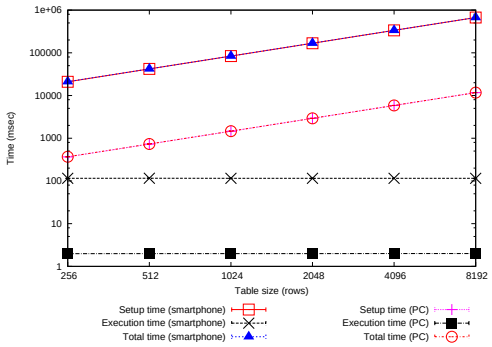

Figure 8: Themis' performance for the millionaires' problem application for different table sizes and constant key size (1024 bit) in the PC and the smartphone. Increasing the table size also increases the setup time but has no effect on the execution time. Also notice that the total time and the setup time are very similar for both platforms.

those of Fairplay. We will investigate the relative performance of these protocols in the next section.

6. PERFORMANCE EVALUATION

In this section we analyze the performance of Themis using two applications: the Millionaires' problem (matching) and the Coffeebucks application (optimization). Both applications were implemented using C language. For additive homomorphism, we used the Paillier Library[4] and for multiplicative homomorphism we used our own implementation of ElGamal cryptographic functions. For comparison purposes, we also implemented these two applications using the Fairplay (SFE) framework. All the source code and supporting material used in our experiments can be downloaded from the following location: `http://www.cc.gatech.edu/grads/c/camrutk/themis`

Our evaluation used two platforms: a desktop PC and a smartphone. The desktop PC has a dualcore 2.50 GHz x86 processor, 2 GB of memory and Ubuntu 9.0.4 (Linux kernel 2.6.28) operating system. The smartphone is an Android G1 phone with a 528 MHz ARM processor, 98 MB of memory and Android 1.6 (Linux Kernel 2.6.29) operating system. For testing in the smartphone, the Themis-based applications were cross-compiled from x86 to ARM using the toolchain included in the Android source code. Fairplay-based applications were first compiled in a desktop PC to create the applications' circuit files. These circuit files and the Fairplay Java code were ported to the smartphone to test performance.

During our performance evaluation, each test was executed at least 10 times to ensure the soundness of the results. We use average values in our analysis and a 95% confidence interval is provided in most of the graphs. Note that these bounds are often difficult to observe in our graphs as the values are very close to the mean.

Our first test evaluates the processing time of the Themis-based millionaires' problem using different key sizes: 512, 768, 1024 and 2048 bit ElGamal keys and a constant table size of 500 rows. The results for the PC and smartphone are presented in Figure 7. As expected, the use of larger key sizes increases the setup and execution time of the application in both platforms (exponential increase). The setup is the most time consuming operation because it involves n public key encryption operations, where n is the size of the query table (e.g.,Q_A). On the other hand, the execution time is significantly faster than the setup time because the former only requires 2 public key encryption operations for any key size. For example, for 1024 bit key, the setup time is 41.01 s and the execution time is 115.58 ms in the smartphone. In the PC, the operations are faster but the trend is similar: 4.76 s and 13.43 ms for setup and execution respectively. These results explain why the total time (setup + execution time) is very close to the setup time (both lines overlap in Figure 7 for both platforms). However, the cost of the setup time can be avoided if the encryption operations are performed in advance (e.g., generate and store the encrypted values periodically). As a result, precomputing the encryption values used during setup will reduce the time required by the application considerably. With this optimization, it is possible to solve the millionaires' problem in mobile devices efficiently (only execution time, $\approx 115ms$).

In the next test, we vary the table size (256, 512, 768, 1024, 2048, 4096 and 8192 rows) while keeping the same key size (1024 bit) for the millionaires' problem application. Figure 8 depicts the results for the PC and the smartphone. As in the previous tests, the setup time is significantly larger than the execution time for both platforms. Similarly, the setup time and the total time lines overlap in this figure for both platforms too. However, while the setup time also grows exponentially with larger table sizes, the execution time remains constant. The reason is that for the setup time, a larger table represents a larger number of public key encryption operations, while for the execution time, only two public key encryption operations are required independent of the table size.

Pre-computing the ciphertexts required during setup operations to optimize performance is not a difficult requirement. For example, a smartphone can generate the encrypted values (e.g., encryption of 1's and 2's) while it is being charged everyday. Therefore, the user will not notice changes in energy consumption or CPU utilization. In addition, smartphone's memory will not be affected significantly. In our implementation, each encrypted value occupies 24 bytes of memory space. Therefore, for a table of size 500, approximately 12 KBytes of memory will be required. Because this is the biggest data structure in the application, we can assume that this is approximately the memory that our application will require. Therefore, we can conclude that the memory requirements of Themis are modest.

In Figure 9 we compare the Themis implementation of the millionaires problem with the Fairplay implementation for different bit sizes inputs (4,6,8 and 10 bits inputs) in the smartphone. The lines in Figure 9 represent the execution time for each implementation (Themis, Fairplay). The Figure clearly shows that the execution time of the Fairplay implementation increases with the number of input bits, while it remains constant for the Themis implementa-

[4] `http://acsc.cs.utexas.edu/libpaillier/`

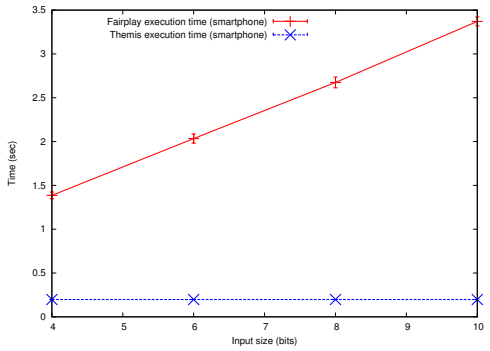

Figure 9: Comparison of Fairplay and Themis (1024 bit key) execution time for the millionaires' problem application in the smartphone. Increasing the input size increases Fairplay execution time but has no effect on Themis execution time.

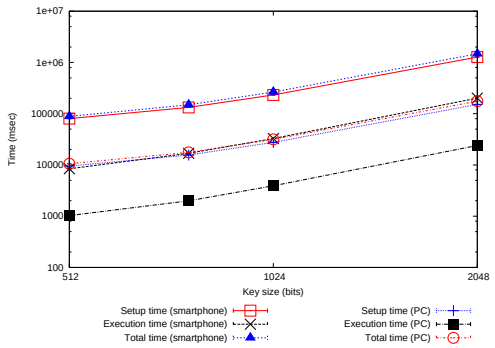

Figure 10: Themis's performance for the Coffeebucks application for different key sizes and constant table size (600 rows) in the PC and the smartphone. Coffeebucks is a more complex application that requires higher setup and execution times than the millionaires' problem. However the trends are similar.

tion. The use of larger number of bits in the input requires larger circuit sizes for Fairplay and bigger tables for Themis. However, as explained before, the table size does not affect the execution time in Themis (only the setup time). Thanks to the mapping function in Themis, we can map different range of inputs without requiring larger table sizes (only granularity is affected). However, if bigger tables are required, the execution time is not affected.

We run similar tests using the Coffeebucks application, a more complex application that performs optimization in addition to matching operations. The only difference with the previous tests is that we use a different range of table sizes: 300, 600, 900 and 1200 rows. Each row corresponds to a *five* minute time interval. Each Coffeebucks location has 12 subentries (rows) for an hour. The results are shown in Figures 10 and 11 for different key sizes and different tables sizes experiments respectively. Overall, the behavior is similar to what we observed in the millionaire's problem evaluation: increasing the key or the table size causes an exponential grow in the time required by the application. Also, the difference between the setup and execution time is smaller but still considerable. We can observe that with larger table sizes, the execution time now also grows exponentially (it was constant for the millionaires' problem). The reason is that the Coffeebucks execution operations are more complex (For example, table operations) than the operations required by the millionaires' problem. Finally, even for this complex application the execution time is acceptable in a mobile

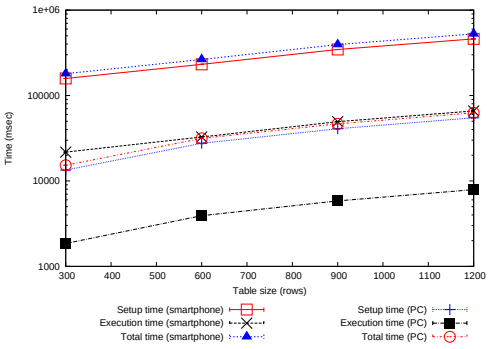

Figure 11: Themis' performance for the Coffeebucks application for different table sizes and constant key size (1024 bit) in the PC and the smartphone. In this test, the execution time also increases with larger table sizes.

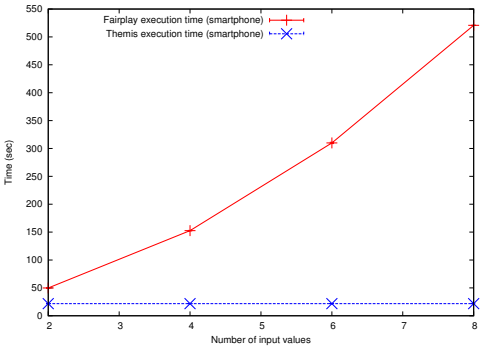

Figure 12: Comparison of Fairplay and Themis (1024 bit key) execution time for the optimization application in the smartphone. Even for very small inputs on a slightly complex problem, Fairplay's execution time increases exponentially.

device: approximately 32.68 sec for a table size of 600 and 1024 bits public key encryption. These results show how Themis allows the implementation of efficient privacy preserving applications that can even run in devices with limited hardware resources such as smart phones.

Our final test evaluates the performance of the Coffeebucks application with Fairplay. The results are presented in Figure 12. As already established, the Fairplay circuits become considerably complex even with a slight increase in the complexity of the application. The graph shows the execution times in seconds for Fairplay and Themis for extremely small input values of 2 to 8. The input values correspond to the number of candidate Coffeebucks locations in the application. For 8 inputs values, Fairplay takes 520.75 s as compared to 21.76 s taken by Themis. The constant time 21.76 s is for input value of 25 which is much greater than 8. Therefore, Themis improves the performance by 96% which proves that it is more appropriate for complex context sensitive applications.

7. CONCLUSION

With the proliferation of computing in the physical world, context aware applications are becoming popular. However, one of the biggest hurdles in fully exploiting the functionalities of such applications is the challenge of providing user privacy. In this paper, we designed and developed *Themis*, a framework that offers user privacy guarantees in two-party context aware applications. Our

framework substitutes private search in place of computationally intensive oblivious communication protocols. The security guarantees provided by Themis are comparable to the existing oblivious communication protocols such as Fairplay, while maintaining better performance on hardware constrained devices. We built and characterized two real world applications using our framework to demonstrate its practical validity. We proved that replacing computation with search is not only feasible but also more efficient in terms of performance. In conclusion, Themis is more appropriate for privacy preserving context aware applications on hardware constrained devices as compared to existing cryptographic solutions available in this space.

Acknowledgments

This work was supported in part by the US National Science Foundation (CCF-0916031). Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

[1] A. Arabo, Q. Shi, and M. Merabti. Towards a context-aware identity management in mobile ad hoc networks (immanets). *Advanced Information Networking and Applications Workshops, International Conference on*, 0:588–594, 2009.

[2] B. Bamba, L. Liu, P. Pesti, and T. Wang. Supporting anonymous location queries in mobile environments with privacygrid. In J. Huai, R. Chen, H.-W. Hon, Y. Liu, W.-Y. Ma, A. Tomkins, and X. Zhang, editors, *WWW*, pages 237–246. ACM, 2008.

[3] M. Bauer, C. Becker, and K. Rothermel. Location models from the perspective of context-aware applications and mobile ad hoc networks. *Personal Ubiquitous Comput.*, 6(5-6):322–328, 2002.

[4] F. Bergenti, C. Caceres, A. Fernandez, N. FrÃűhlich, H. Helin, O. Keller, A. Kinnunen, M. Klusch, H. Laamanen, A. Lopes, S. Ossowski, H. Schuldt, and M. Schumacher. Context-aware Service Coordination for Mobile e-Health Applications. In *Proceedings of the European Conference on EHealth (ECEH06)*, volume P-91. GI-Edition, 2006.

[5] J. Bethencourt, D. X. Song, and B. Waters. New techniques for private stream searching. *ACM Trans. Inf. Syst. Secur*, 12(3), 2009.

[6] G. Chen and D. Kotz. A survey of context-aware mobile computing research. Technical report, Hanover, NH, USA, 2000.

[7] I. Y. Chen, S. J. Yang, and J. Zhang. Ubiquitous provision of context aware web services. *Services Computing, IEEE International Conference on*, 0:60–68, 2006.

[8] A. K. Dey, G. D. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, 2001.

[9] J. C. Freytag. Privacy in location-aware systems. *SIGSPATIAL Special*, 1(2):4–8, 2009.

[10] W. Gasarch. A survey on private information retrieval. *Bulletin of the EATCS*, 82:72–107, 2004.

[11] B. Gedik and L. Liu. Location privacy in mobile systems: A personalized anonymization model. In *ICDCS*, pages 620–629. IEEE Computer Society, 2005.

[12] B. Gedik and L. Liu. Protecting location privacy with personalized k-anonymity: Architecture and algorithms. *IEEE Trans. Mob. Comput*, 7(1):1–18, 2008.

[13] B. G. Gedik and L. Liu. A customizable k-anonymity model for protecting location privacy. Technical report, Apr. 08 2004.

[14] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *SFCS '86: Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, pages 174–187, Washington, DC, USA, 1986. IEEE Computer Society.

[15] Google, Inc. Google Latitude. `http://www.google.com/latitude/intro.html`, 2010.

[16] A. Kapadia, N. Tri, C. Cornelius, D. Peebles, and D. Kotz. Anonysense: Opportunistic and privacy-preserving context collection. In *In Proc. of 6th IntÕl Conf. on Pervasive Computing*, 2008.

[17] A. Khoshgozaran, H. Shirani-Mehr, and C. Shahabi. Spiral: A scalable private information retrieval approach to location privacy. In *Mobile Data Management Workshops, 2008. MDMW 2008. Ninth International Conference on*, pages 55–62, April 2008.

[18] D. Kifer and J. Gehrke. l-diversity: Privacy beyond k-anonymity. In *In ICDE*, page 24, 2006.

[19] L. Kruger, S. Jha, E.-J. Goh, and D. Boneh. Secure function evaluation with ordered binary decision diagrams. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 410–420, New York, NY, USA, 2006. ACM.

[20] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *In Proc. of the 38th Annu. IEEE Symp. on Foundations of Computer Science*, pages 364–373, 1997.

[21] D. Lopez-de Ipia, J. Vazquez, and J. Abaitua. A context-aware mobile mash-up platform for ubiquitous web. In *Intelligent Environments, 2007. IE 07. 3rd IET International Conference on*, pages 116–123, Sept. 2007.

[22] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay—a secure two-party computation system. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 20–20, Berkeley, CA, USA, 2004. USENIX Association.

[23] D. K. Mishra and M. Chandwani. Arithmetic cryptography protocol for secure multi-party computation. In *SoutheastCon, 2007. Proceedings. IEEE*, pages 22–22, March 2007.

[24] T. Nakamura, S. Inenaga, D. Ikeda, K. Baba, and H. Yasuura. Anonymous authentication systems based on private information retrieval. In *Networked Digital Technologies, 2009. NDT '09. First International Conference on*, pages 53–58, July 2009.

[25] R. Ostrovsky and W. E. S. III. Private searching on streaming data. Cryptology ePrint Archive, Report 2005/242, 2005. `http://eprint.iacr.org/`.

[26] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. pages 223–238. Springer-Verlag, 1999.

[27] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Security and Privacy, 2000. SandP 2000. Proceedings. 2000 IEEE Symposium on*, pages 44–55, 2000.

[28] M. Srivatsa, A. Iyengar, J. Yin, and L. Liu. A scalable method for access control in location-based broadcast services. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 256–260, April 2008.

[29] A. C. Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS '08. 23rd Annual Symposium on*, pages 160–164, Nov. 1982.

APPENDIX

A. PALLIER CRYPTOSYSTEM

The paillier cryptosystem [26] is a probabilistic asymmetric algorithm for public key cryptographic. The cryptosystem is based on the hypothesis that computing n-th residue classes is difficult. Informally this means that given a composite n and an integer z, it is hard to decide whether z is a n-th residue modulo n or not. i.e. Whether there exists a y such that z=y^n mod n^2.

A.1 Paillier Algorithm

In this section, we will give more details on the functionalities of the Paillier algorithm.

Key Generation.

1. Choose two large primes p and q such that gcd(pq,(p-1)(q-1)) = 1. If both p and q are of the same length, this property is assured. []
2. Compute n=pq. (p-1).(q-1) is called the Euler's totient ($\phi(n)$) of n. The Carmichael's function on n is defined as λ = lcm(p-1,q-1)
3. Select a random integer g where g $\in Z^{\star}_{n^2}$
4. Check the existence of modular multiplicative inverse : μ = (L (g^{λ} mod n^2))$^{-1}$ mod n. where function L is defined as : L(x) = $\frac{x-1}{n}$
   This check ensures that n divides the order of g.
5. (n,g) serve as the public parameters. The private key is (λ, μ) which depend on (p,q).

Encryption.

1. Let m denote the plaintext to be encrypted, m $\in Z_n$.
2. Select a random element r $\in Z_n^\star$
3. Ciphertext c = $g^m.r^n \bmod n^2$.

Decryption.

1. Ciphertext c $\in Z_{n^2}^\star$
2. Plaintext m = L($c^\lambda \bmod n^2$) . μ mod n

Homomorphic properties of Paillier cryptosystem.

The Paillier encryption functions are additively homomorphic. It leads to the following properties :

1. The product of two ciphertexts gives the sum of their plaintexts on decryption.
   D(E(m1,r1) . E(m2,r2) mod n^2) = (m1 + m2) mod n.
2. The product of a ciphertext with a plaintext raised to g gives the sum of their plaintexts on decryption.
   D(E(m1,r1) . g^{m2} mod n^2) = (m1+m2) mod n.
3. An encrypted message raised to a constant k gives the product of their plaintexts on decryption.
   D($E(m1, r1)^k$ mod n^2) = (k.m1) mod n.
4. An encrypted plaintext raised to the power of another plaintext gives the product of the two plaintexts on decryption.
   D($E(m1, r1)^{m2}$ mod n^2) = (m1.m2) mod n.

A.2 Security of Paillier cryptosystem

The Paillier cryptosystem provides security against chosen-plaintext attacks (IND-CPA). Because of the above mentioned homomorphic properties of the system, it is susceptible to adaptive chosen-ciphertext attacks (IND-CCA2). Though having malleability is not good in cryptographic research, these homomorphic properties can be leveraged in a number of applications. An improved cryptosystem that incorporates the combined hashing of message m with random r is IND-CCA2 secure.

B. APPLICATIONS

The Themis framework is highly extensible and can be adapted for a wide range of context sensitive applications. We explain two such applications in detail in this section.

B.1 Setting an Appointment

A student and a professor attempting to establish a meeting time wish to do so without revealing their entire calendars. The optimization protocol can be adapted for this application. The student generates a blind query containing preferences for his free times on a particular date and time. The ciphertext values in the student's query are his preferences for each free time slot in his calendar and *zero* at unavailable slots in the query. The professor generates his own table of free times and their respective preferences and runs the student's query against these values.

The professor adds some noise entries to the result table and shuffles the result. He then sends the result with his corresponding preferences for each slot matched. The student decrypts the result to get a table containing integer values and finds the optimal solution by finding the entry which has the highest addition and has least difference between the preferences. The student sends the top x solutions back to the professor ($1 \leq x \leq size(result)$). The professor can verify that the top slot was indeed one of his selections and approve the meeting.

B.2 Medical Alert Service

Existing medical alert services inform loved ones of the person in need, in case of an emergency. However, physical distance may preclude specific loved ones from helping. We design a protocol using Themis for medical services that does not expose the locations of the participants.

Our medical alert application selects the person to contact based on the their physical location and their status within the sender's community of interest (i.e., the frequency with which calls are exchanged). The provider and Alice's loved ones willingly participate in the application. The telephony provider holds records of all the customers' locations and their talk times with the people on Alice's contact list. The medical alert server maintains a white-list of people on Alice's contact list, with whom she is willing to share her medical emergency information. When in need of medical aid, Alice sends a request to the medical alert server through her cell phone. The medical alert server then creates blind queries containing the names of people on the white-list and some conditions such as the potential helpers should reach Alice in less than 15 minutes.

The provider allows the medical server to execute blind queries only on Alice's records in the billing and location database. Using private search, the medical server can obtain the list of people who belong to the white list and are *geographically and personally* close to Alice. The telephony provider hides the sensitive information of these matches from the application server using the noise addition procedure. It sends the garbled results back to the medical alert server for decryption and intermediate optimal solution generation. The medical server sends the intermediate optimal solution to telephony service provider, who in turn sends alert messages to the subset of people satisfying all the conditions of optimality.